Ukraine Conflict Similar Tweets

Gabriele Cerizza

Università degli Studi di Milano
gabriele.cerizza@studenti.unimi.it
https://github.com/gabrielecerizza/amd_project

1 Introduction

In this report we detail our findings in the study of an algorithm capable of identifying similar pairs of documents within massive datasets. The experiments illustrated hereinafter were carried out as part of a project for the Algorithms for Massive Datasets course of Università degli Studi di Milano.

2 Dataset

The dataset employed in our experiments was the "Ukraine Conflict Twitter Dataset" from Kaggle[1], released under the CC BY-NC-SA 4.0 license. This dataset boasts a total of over 40 million tweets concerning the conflict between Russia and Ukraine, which broke out on the 24th of February 2022. Those tweets were collected daily since the same date by monitoring hashtags. The dataset was first published on the 27th of February 2022. In the remainder of this report, we will refer to the version 127 of the dataset, downloaded on the 19th of June 2022.

The dataset comprises 109 compressed CSV files. For each sampled tweet, we can find information concerning the author, the text, the hashtags, the language and the date of creation of the messages. It is worth noting that the naming of the files is not consistent, which hinders attempts to process the tweets chronologically. (Image of a tweet in dataframe?)

2.1 Filtering and Preprocessing

At the outset, within each given file, we dropped duplicate tweets, which were present in a sizeable number. Beforehand, we stripped all URLs from the tweets, since we found some that differed only in a URL placed inside the text.

A number of preprocessing steps were then performed on the documents, having two objectives in mind: (i) discarding noisy and irrelevant information; and (ii) reducing the number of different characters that could be found in the tweets, which adversely affects the model complexity (see further ...).

In particular, we performed the following preprocessing steps:

[1] www.kaggle.com/datasets/bwandowando/ukraine-russian-crisis-twitter-dataset-1-2-m-rows

1. we replaced words with accents with their counterparts without accents;
2. we replaced special UNICODE characters with their ASCII counterparts, for instance "ℝ" with "R";
3. we replaced the ampersand with "and";
4. we replaced each token with the corresponding lemma, to increase the match between words like "invasion" and "invaded";
5. we removed stop words;
6. we converted everything to lowercase;
7. we removed punctuation symbols;
8. we removed non-ASCII characters, except for cyrillic characters;
9. we normalized the whitespace.

While numbers might be considered noise in some contexts, here we decided to keep them. Indeed, in our context, numbers could be found in dates, Twitter handles of influential people and military equipment (e.g., the Russian T-72 tank or the M982 Excalibur 155 mm shell) and could help in discriminating the documents.

When feeding the Transformer-based model, which was used for comparison, we performed only light preprocessing. We replaced the ampersand, we lowercased the text and we normalized the whitespace. In this way, we could leverage the ability of the Transformer to exploit the context of each word, which would have been hampered by removing punctuation and stop words or by lemmatizing the tokens.

3 Models for Document Similarity

In this section we briefly describe the approach used to find similar documents (Sec. 3.1) and the neural network model we used as a baseline (Sec. 3.2).

3.1 Locality-sensitive hashing technique

In order to find the pairs of similar documents, we employed the algorithm described in [1], which can scale up to massive datasets. This algorithm converts each document to a set of k-grams (shingles), builds a characteristic matrix and then a signature matrix by using hash functions, applies a locality-sensitive hashing (LSH) technique to find candidate similar pairs and, finally, checks the candidate pairs against the signature matrix to discard false positive pairs. Optionally, one could check the candidate pairs also against the characteristic matrix, if available.

Concerning the implementation details, we first discuss the various hash functions exploited by the algorithm. To avoid keeping the shingles in main memory, the algorithm hashes each shingle to a bucket, in the form of an integer. The total number of buckets associated to the hash function has a significant impact on the execution time. Being confronted with the problem of finding hash functions that could map to a different number of buckets, we decided to generate

different hash functions by truncating the output of the SHA-256 hash function at varying length of bits. In this way, we could experiment with several number of buckets.

The algorithm requires also hash functions that could map a row index to another row index. These hash functions are used to efficiently perform permutations of the rows. Since the number of these hash functions is a hyper-parameter set by the user, we needed a way to generate an arbitrary number of different hash functions. To this end, we followed the approach described in [2] to generate hash functions of the form

$$h(x) = (ax + b) \bmod c \ ,$$

where x is a row number, a and b are random numbers smaller than the maximum row number and c is a prime number higher than the maximum row number. Note that a and b must be unique for a given signature matrix.

In order to build the signature matrix, the algorithm also requires a characteristic matrix, where each column represents a document and each entry indicates whether or not a given shingle is contained in a given document. We store the characteristic matrix as a dictionary that maps each document index to its set of shingles. In this way, we do not waste memory to store the zeroes.

Concerning the LSH technique, we simply note that, to check which columns are equal,

Optimizer to find b and r Hash in LSH Last layer has most meaning TFIDF? Need to traverse the whole dataset to compute, taxing operation Two semantically different pipelines: to filter almost identical tweets, and the other. Punctuation and non-ascii removed later, because they might help the lemmatizer CHANGED PIPELINE (some differed for only some whitespace or for a capital letter (mint and MINT))

Problem with low threshold, ends RAM with combinations As such, we decided to experiment with The algorithm used dictionary to store shingles

Strip again punctuation because some of the lemmas had punctuation symbols like _ in them.

fast spike increase in shingles at low documents

used spacy

Cannot use spacy on dataframe, takes too much time

crash with t.02, too many pairs with t=0.4 6346167 candidate pairs, actual 342671

We do not drop duplicates across different files, as that would require to load in main memory millions of documents. Plenty of retweets. Plenty of very short tweets that are identical after preprocessing /lemmatization. But filtering on string length isn't suitable due to the instrinsic short nature of tweets. Sometimes only changed a handle or more, but removing handles might remove important parts (many quote zelensky with handle for instance).

Removing handles might be somehow questionable... Si ma poi se tanto retweettano in altri file compare comunque.

Dai, remove handle e testi più lunghi di tot... ONLY EN language Should have kept the retweet flag

CHECK TP ON ACTUAL OUTPUT, AFTER THRESH CHECK

Generate the combinations manually and add one at a time, to avoid ram problem?

3.2 Transformer neural network

The dataset did not provide a ground truth against which we could compare our results. Therefore, we decided to use a state-of-the-art model in the field of natural language processing (NLP) to serve as a baseline. To this end, we opted to use MPNet, which is a Transformer-based model based on the revolutionary BERT neural network.

Why better than word2vec

A neural network would not be able to compute the pairwise similarity between each pair of documents in a reasonable amount of time. As a consequence,

After feeding the documents to

Example of similar tweets Non-cryptographic function

and which has been found to work well Link a zip with checkpoints in the github To achieve this goal To this end

, so we removed all URLs beforehand. We also removed URLs from tweets, since that differed only in hashtags or in placed within the messages were not filtered, since .

might skew

each of which organized in fields providing information about the author each of which

foregoing

Not actually deduped.

3.3 Conclusion

With regard to the paragraph classification task, our MTL model outperforms the baseline methods. Further improvement can be attained with a more fine-grained analysis of the Wikidata properties of each Wikipedia page.

Our event model struggles to correctly identify trigger words. One reason is that the vast majority of the spans do not contain events and, therefore, finding the correct event span and the correct event type becomes a highly imbalanced problem. The EventGen model fares better and the predictions show that in many cases the mistakes were semantically close to the gold labels. Our argument model compares favourably with the baseline methods. We could improve the model by appending argument type constraints to the templates.

References

1. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, 3 edn. (2020). https://doi.org/10.1017/9781108684163
2. Liu, E.: Text Similarity. http://ethen8181.github.io/machine-learning/clustering_old/text_similarity/text_similarity.html (2015), [Online; accessed 23-June-2022]